立体Pentominoのプログラム (計算機によるゲーム・パズルの具体化の検討)

AUTHOR(S):

竹内, 郁雄

CITATION:

竹内, 郁雄. 立体Pentominoのプログラム (計算機によるゲーム・パズルの具体化の検討). 数理解析研究所講究録 1974, 217: 36-56

ISSUE DATE:

1974-07

URL:

http://hdl.handle.net/2433/105276

数理解析研究所講究録
217巻 1974年 36-56

立体pentominoのプログラム

電電公社武蔵野電気通信研究所
竹内 郁雄

1. はじめに

"GPCC發足に類するを試みんと思い立ったのは，昨今のことではなけれど，實際その期熟するは忘れもせない，立体pentomino六百六拾時間たる見積りの世に問はれたる*)翌日の本年一月の拾一日で，その日以来本来の業務はあり，殊には不肖私も承軽症の風邪に犯されたれど（中略），難題が起ると態と之を打棄ておき，厠上又褥上にて脳漿を絞りしことも常であったが（中略），到々拾一時間強にて全数解を求めたるものなり**)"というわけで，当初二桁時間であればという期待を越えて，一桁時間に肉迫する結果が出たのでここに報告しよう．

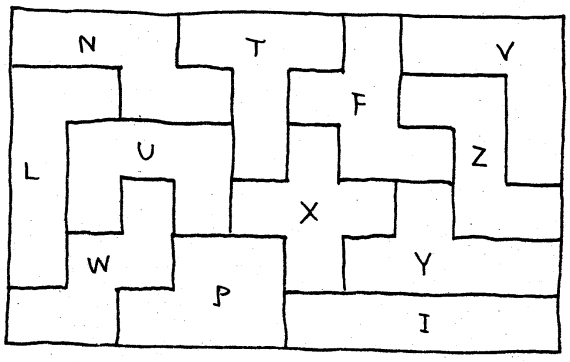
---

*) 航宇研の磯部氏「立体ペントミノの解法について」第15回プログラムシンポジウム 1974.1

**) 尊敬する松浦政泰氏の「世界遊戯法大全」による．

2. 立体 pentomino について

単位立方体を5個平面上に並べると次のような12種類のピースが出来る。(紙面節約のために6×10のお盆に最密充填した形で示す。)



さて，これを3×4×5の直方体スペースに充填しようというのが，ここでいう(狭義の)立体pentominoである。これが解を持つことは古くから知られていたようであるが，その解の総数が知られたのは比較的最近である。我が国の天洋で売り出したプラスチック製の製品には，FACOMという字が箱にしたためてあるが，その横に"460通りの組合せができる3次元の立体パズル!!"と丁寧にニつもビックリマークがついている。しかし，実は解の総数は1967年にC.J. Bouwkampという人が求めた3940通りというのが正しい。使った計算機はIBM-1620でFORTRANを言語としたらしいが，そのときに要した時間は"several hundreds hours"だったそうである。ただし，若干のコメントがあって，CDC-3600上のアセンブラ・プログラムの集積を利用すればる時間位で全数解が求まるであろうと言っている。しかし，それがその後実

現されたという話は伝わっていないようである．磯部氏はかなり一般的な構想のもとで（4次元直方体などを含む）プログラムを作られたようであるが，HITAC 5020Fで660時間かかるという見積りを出されたのは，IBM-1620との単純な比較だけでもやや過過ぎるのではないかと思われる．CDC-3600程度の機械は現在ミニコンとしてかなり出まわっているので，3時間が本当かどうかの意味も含めて，時間短縮を試みることにしよう．

3. プログラムを組む上での基本方針と構想

使用する計算機は現在かなり世に出ている pdp 11/20である．この機械の基本的な特徴を必要なだけ述べておくと

a. レジスタが8個あり，1つはプログラム・カウンタで1つはサブルーチン等のためのスタック用レジスタである．（前者をPC，後者をSPと呼ぶ）あとのレジスタは便宜上R0，R1，…，R5と名付けておく．これら8個のレジスタに演算上の機能の差はないし，すべてインデックス・レジスタとして使える．

b. 演算速度．レジスタ．レジスタで加算が2.3μsec，メモリを1回アクセスすると1.5μsec～3.9μsec余分にかかる．いわゆる中速度の計算機である．

c. バイトでアドレスがついている. 1語は16bitである.

d. シフト関係の命令が弱いのでビット処理を得意としない.

また, 我々の使用したシステムに関してさらにつけ加えると

e. ディスク・オペレーティング・システムで28K語のメモリを持ち, 他の周辺機器(MT, DISK…)もかなり完備している.

f. アセンブラはマクロの入れ子が何重にも出来る機能を持つなどかなり高度なものが用意されている.

さて, 立体pentominoのプログラム作成の基本方針としては

(1) シフトが弱いので, bit単位でマス目情報を保持せずに, word単位で保持する. つまり通常のarrayで直方体を表現する.

(2) 番号の若いマス目から, 順にピースを埋めていく. 磯部氏のプログラムはピースをバラバラに入れていくのだが, こちらの方法のほうが(単純なことだが)10倍位速くなる. ピースの順番は, 大体普通.

(3) コーディング技術で時間を稼ぐ. 即ち複雑なヒューリ

スティックを入れない．だから原理的にシラミつぶしプログラムとする．

(4) ステップ数，メモリ使用量については湯水の如きの方針を貫く．

(5) 対称解は最初から除くようにする．(プログラムの信頼性はすこし下がるが気にしない．)

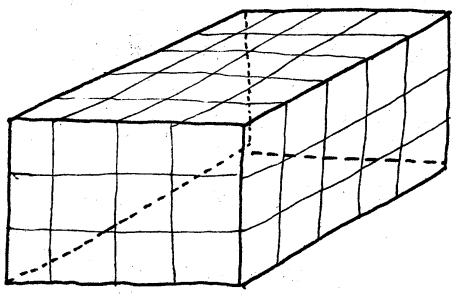
(6) (4)に関連するが，ループ管理のためのoverheadがもったいないので，ループを廃止しベタ書をする．またプログラム・フェッチとデータ・フェッチを一体化して時間を節約する．すなわちデータをプログラム中に本質的に埋め込んでしまう．(これによって2～4倍のスピードアップがはかれる．)

4. プログラムの基本骨格

3×4×5の直方体は次の様に表現される．(あとでアセンブラに密着した説明をするので，予備説明としてリスティングをのせておく) arrayの各語はマス目か，境界を示すためのダミーであり，前者はそこが空いているときゆ，なにかピースがつまっているときには，そのピースの名のコード(ASCII)が入れられている．後者は常に1がはいっている．次のリストはarrayの初期状態である．

```
12                      .MACRO  TBGNR N
13                              .IF EQ,N
14                              1,0,0,0,0
15                              1,0,0,0,0
16                              1,0,0,0,0
17                              .IFF
18                              1,1,1,1,1
19                              1,1,1,1,1
20                              1,1,1,1,1
21                              .IFTF
22                              1,1,1,1,1
23                              .ENDC
24                      .ENDM
25 00002                        TBGNR 1
26 00052                TB:     TBGNR 0
27 00122                        TBGNR 0
28 00172                        TBGNR 0
29 00242                        TBGNR 0
30 00312                        TBGNR 0
31 00362                        TBGNR 1
```

引数Nなるマクロ TBGNRの定義
N=0なら以下を..
条件が下(i.e. N≠0)なら以下を..
条件が下でも下でも以下を..
条件文終り
マクロ定義終り
array (代表マスをTBと呼ぶ)



マス目の番号はTBから数えて何番地(8進で表示, 以後断わりなき数値は8進数)であるかで表わされる. 左図で1番奥の3×4平面は下の(a)のように番号づけられており, その手前の断面は左の(b)の如くなっている. マスが一つ手前だと50たされた番号になることに注意しておこう.

2	4	6	10
14	16	20	22
26	30	32	34

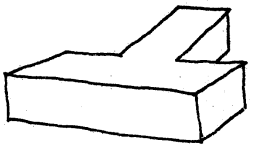
(a)

52	54	56	60
64	66	70	72
76	100	102	104

(b)

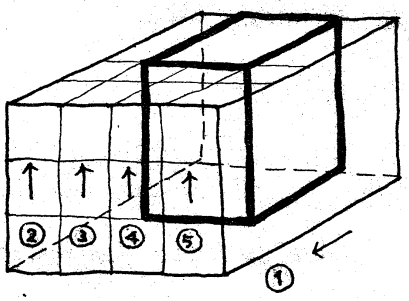
ピースはおのおの何種類かの向きを持つ. これを野下氏の用語に従ってスタイルと呼ぶことにしよう. 12枚のピースのスタイルで上の直方体に入るようなものは全部で160種類あ

る．スタイルは，一番若い番号に入るべき単位立方体から他の4つの立方体への番号変位を書き並べることによって表現出来る．たとえば，Tというピースの(c)なるスタイルは


(c)

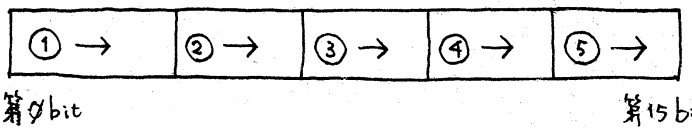
5Φ, 12Φ, 116, 122

というふうに表現出来る．番号変位の順番は都合のよいように選んでよい．さて，あるスタイルを持ってきたとき，それが直方体の任意の番号のマス目を基点として入り得るわけではない．実際，上例の(c)なるスタイルは直方体の左端，右端，前から1番目と2番目の3×4の断面のマスを基点として入ることが出来ない．(要するに案外入りにくい) これを各スタイルについて最初に調べておく方が得策なので，各スタイルについてvalid space (vsp) なるデータを用意しておく．上の例では



左の太枠のような部分直方体が表わしたいものである．処理を速くするために，これを1語で表現する．直方体は12本のI(長さ1の棒)からなると考えられる．この棒1本1本に1bit対応させる．またこれに直交する3×4平面が5枚あるが，このうち一番奥の(番号の若い)平面を除いたものに各々1bit対応させる．対応のさせ方は上図と次頁の図をにらんでいただけ

ればア解いただけよう.



ピタリ16bit. しかも8進数表示と図示表示とのマッチが非常によい。さてbit情報の意味は

{ 0: その棒または平面に部分直方体が交わる.
  1: 〃 が交わらない.

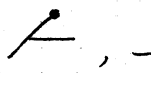
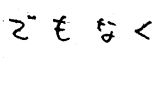
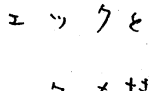
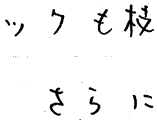
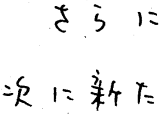
結局, 上の例は, VSPが37007となる. (一番奥の平面はどんなスタイルでも最低2ヶ所入りうるので, 対応するbitは常に0となる. 幸い第-1bitは常に0と考えてよい.)

あるマス目が考えているスタイルの墓穴となりうるかどうかのチェックをするために, 直方体の各マスに次のようなデータを対応させておく. つまり上のbitマップと同じマップを使い, そのマス目に関係した棒と平面のところにだけ1をたてるのである. (一番奥の平面は平面用のbitが立てられないが支障はない.) たとえば番号54のマス目に対応しては,

1001000 というデータが対応させられる. このデータはプログラムの一番最初で生成され記憶される. チェックはこのデータとVSPのANDをとって, all0 ならOKで, キ0であれば墓穴になり得ないという簡単なものである.

さて，基底チェックがOKだったとすると，次は番号変位に従ってマス目が空いている(=φ)か，否か(≠φ)かを調べればよい．しかるに，あるピースのスタイルを調べているとき，ある番号変位のマス目が塞がっていたら，残りいくつかのスタイルのチェックを自動的にサボってもよいことが往々にしておこる．たとえば，ピース丁で

5φ，12φ，46，44

というスタイルを調べているとき，もし5φなる番号変位のところのマス目が塞っていれば，  ，  ，  ，  ，  などのスタイルは調べるまでもなくだめだということがわかる．これを便宜上，文脈チェックと呼ぶことにしよう．(注意；基底チェックも文脈チェックも枝刈りをしているわけではないことに留意されたい．) さらに5φ，12φについてはOKで，46で駄目とわかれば，次に新たに5φ，12φについて調べる必要もないことに注意しよう．すなわち，あるスタイルで途中で失敗すれば，次のスタイルの途中から調べれば十分なこともある．このような文脈チェックを可能な限り取り入れた速度の向上が期待が出来る．

ここで1つのスタイルに対するチェックを書き下してみることにしよう．R1には当座のマス目のビットパターンがはい

っておリ，R0には当座調べているマス目の番地（TB+番号）がはいっておリ，R5にはピースを実際に埋め込むサブルーチンの番地がはいっている．（飛び先番地をレジスタに持っていると命令語が短縮できる．）；はコメントの意．

```
    ; スタイル・チェック入口
①   BIT   R1, the VSP ; 基点チェック．BITは二つのオペランドのANDをとって条件コードを立てるだけの命令．
②   BNE   上が失敗したときに飛ぶべきスタイル・チェック
          ; BNE ≡ Branch if Not Equal(to zero)
③   TST   番号変位1(R0) ; TST≡TeST．オペランドに関する条件コードをたてるだけの命令．
          オペランド X(レジスタ) の意味は，
          レジスタの内容＋Xが実効番地である．
④   BNE   これが失敗したときの飛び先．
⑤   TST   番号変位2(R0)
⑥   BNE   これが失敗したときの飛び先
⑦   TST   番号変位3(R0)
⑧   BNE   これが失敗したときの飛び先
⑨   TST   番号変位4(R0)
```

⑩　BNE　これが失敗したときの飛び先

⑪　JSR　PC,(R5)　;これは，全部のチェックにパスしたときに実行される．JSR≡Jump SubRoutine．pdp 11 ではサブルーチンジャンプで戻り先が自動的にスタックされる．つまり，次の代替スタイル・チェックの番地が自動的に保存される．

各スタイルにつき以上11ステップ(10進)，16語(10進)のプログラム・セグメントが対応する．これを全部手で書くのは大変だから，マクロを使う．先程の下のスタイルはマクロで

STYLE　VSP2,1,5Φ,12Φ,46,44,7Φ,1ΦΦ,13,13.

STYLE：マクロ名　VSP2：37ΦΦ7につけた名前.　1：基点チェック飛び先情報　5Φ,12Φ,46,44：番号変位　7Φ,1ΦΦ,13,13：文脈チェック情報

と書かれる．基点チェック飛び先情報は，基点チェックに失敗したときに，次のスタイル・チェックをいくつ省略するかに関するものである．(うしろに)同じVSPのスタイルがn個並べばこれはnになる．文脈チェック情報は4つの番号変位に対応して4つある．末桁はスタイル・チェックのどの部分に飛ぶかを表わし，それを除いた残りの桁は基点チェック情報に同じ意味．たとえば上の例では，番号変位5Φで失敗すれば，次に続くスタイル・チェック6つを省略して7つ目に，12Φで失

敗すれば8つ目に，46で失敗すれば1つ目（i.e. 次）のスタイル・チェックの3番目の番号変位のチェックに従べという意味である．（実はこのあとにもっとたくさん省略できるチェックがあるかも知れぬが，条件分岐命令は $\pm 128_{10}$ 語のスキップしか出来ないという制約があったりして高々7ゆ，1ゆゆで押さえられる．）

このように出来る限り省略を行なうのであるが，実際これを手作業でやったためかなり大変であった．たとえば，省略を伸ばすために次のようなことをする．

i	VSP_i	変位$_{i_1}$，変位$_{i_2}$，変位$_{i_3}$，変位$_{i_4}$
j	VSP_j	変位$_{j_1}$，…
k	VSP_k	変位$_{k_1}$，…
ℓ	VSP_ℓ	変位$_{\ell_1}$，…
m	VSP_m	変位$_{m_1}$，

というようにスタイル・チェックが並んでいて，変位$_{i_1}$ = 変位$_{j_1}$ = 変位$_{k_1}$ であり，$VSP_i \subseteq VSP_m$，$VSP_i \cap VSP_\ell = \phi$（空）なら変位$_{i_1}$で失敗したとき，$j$，$k$ なるスタイル・チェックを省略するのはもちろん，$VSP_i \cap VSP_\ell = \phi$（空）により ℓ も省略し（変位$_{i_1}$を調べるためには VSP_i のチェックにパスしているはず），さらに VSP_m の基底チェックも省略するのである．ただし，この省略が連鎖するととんでもないことになる．だから推移条

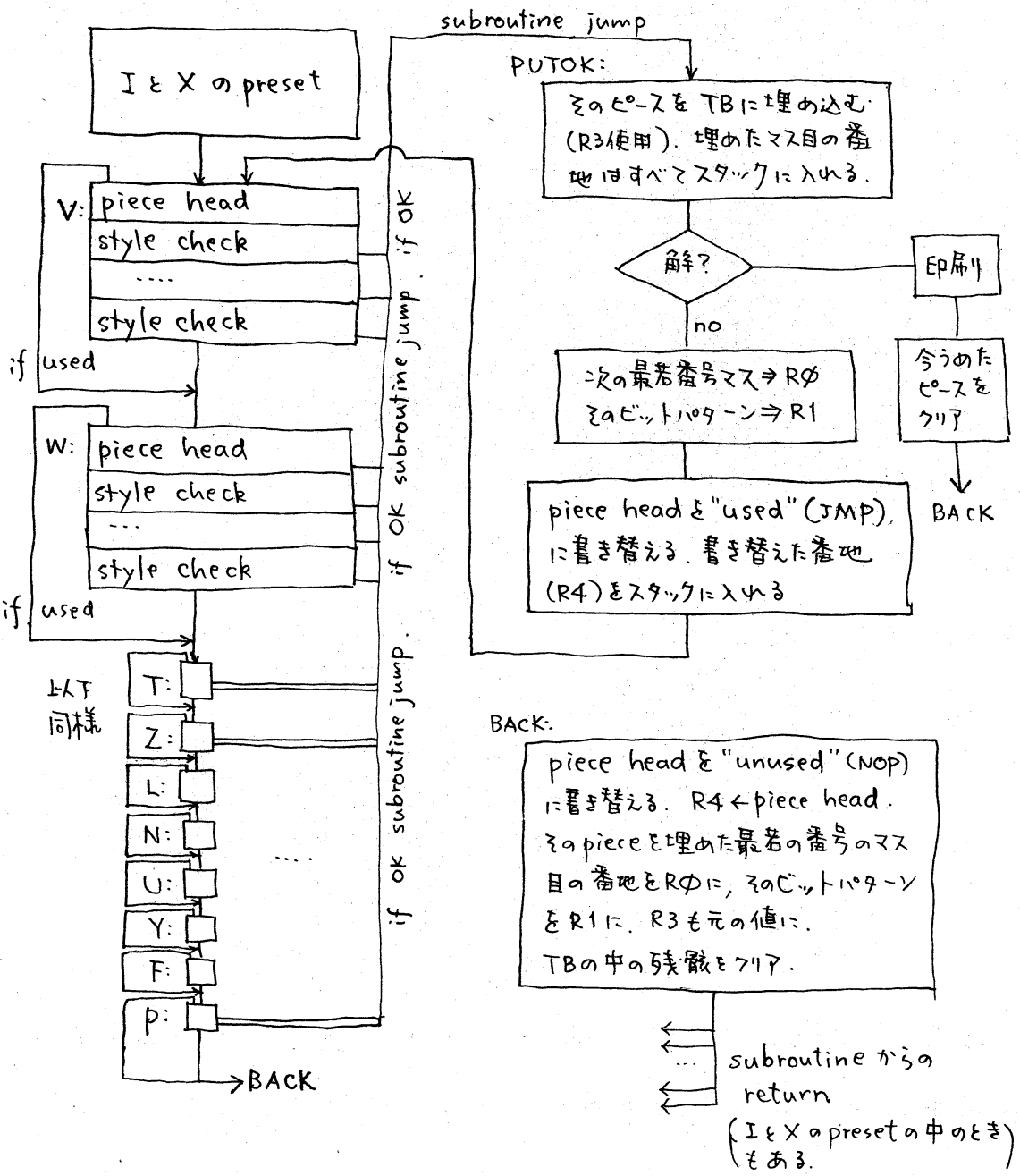
件に常に気を配ったかなり疲れる作業となる.

さて一つのピースのスタイル・チェックはまとめて述べられるが, その先頭に次のような小さなプログラムがつく.

```
piece名 (label):   JMP   次のピース ; 次のピースがなければ
                                  backtrack用routineへ.
                   MOV   piece名のASCIIコード, R3
                                  ; スタイル・チェックをやる
                                  間R3にはピースの名を保持
                                  させておく.
                   MOV   piece名[label], R4
                                  ; このプログラムの一番最初
                                  の命令を書き換えるための
                                  番地をR4に保持しておく.
```

最初のJMP命令はそのピースがすでに使われたかどうかで書き換えられている. すでにそのピースを使ったのであればもうそのピースを調べる必要はないから, ここは上の通りのJMP命令である. もしまだ使っていないのであれば, 以下のステップとスタイル・チェックの列を実行しなければならないから, これは等価的にNo Operationの命令にかえられる. (実際には BR ・+4 , BR≡Branch , ・≡location counter, 上のJMPは2語命令であるから, 1語スキップせよという命令)

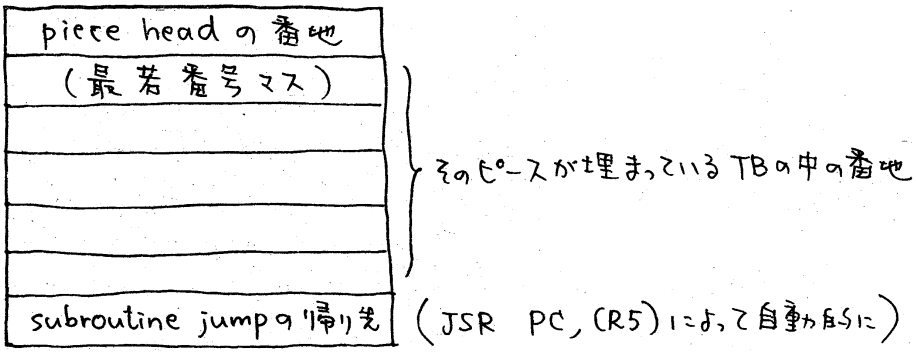
さて以上見たようにこれで，いわゆるスタイル表現用情報，used piece 情報，next style 情報などは完全にプログラム中に埋め込まれたことになる．プログラムの全構成を模式的に示すと（ただし対称解に関する部分は省く）



この図からわかるようにプログラムは大きな再帰的プログラムとなっている。(エとXのpresetの部分はっきりかかなかったが，これとほぼ同じで，BACK routineは他のピースと共用) pdp 11の自動スタックの機能，あるいはレジスタをインデックスとして使ったときの自動インデックス増減機能は，このプログラムの性能に大きく寄与している。たとえばピースをクリアする部分は

```
CLR   @(sp)+
CLR   @(sp)+
CLR   @(sp)+
CLR   @(sp)+
CLR   @(sp)+
```

と本当に5ステップ(5語)ですんでしまう。なおスタックは一本で，1つのピースがはいると次のようなエレメントがスタックの上にのせられることになる。このスタックへのアクセスはSPレジスタを使う。



また，PUTOK routineでTBのマスを埋めるときの番号変位の情報はsubroutine jumpの帰り先からもとのチェックプログ

ラムをアクセスして知る．これが全部で17_{10}ステップ(21_{10}語)かかるのはいたしかたない．ちなみにPUTOK routineは全部で，28_{10}ステップで中に次の最若番号マスを捜す2ステップのループがある．BACK routineは11_{10}ステップである．

次にスタイル・チェックのマクロとpiece headのマクロをリスト通りに示しておく．

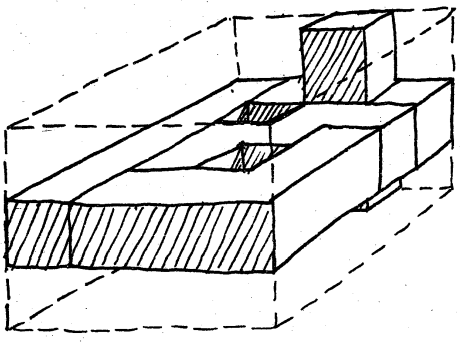
```
.MACRO  STYLE VSP,VSPBR,L1,L2,L3,L4,L1BR,L2BR,L3BR,L4BR
        BIT R1,#VSP
        BNE VSPBR*40-4+.
        BRCNT3=0
.MACRO  CHECK LN,LNBR
        TST LN(R0)
        BRCNT1=LNBR/10
        BRCNT2=40*BRCNT1-12-BRCNT3+.
        BRCNT1=-BRCNT1*10+LNBR
        BNE BRCNT2+<BRCNT1*6>
        BRCNT3=BRCNT3+6
.ENDM
        CHECK L1,L1BR
        CHECK L2,L2BR
        CHECK L3,L3BR
        CHECK L4,L4BR
        JSR PC,(R5)
.ENDM

.MACRO  PIECE IDENT,NEXTP
IDENT:  JMP NEXTP
        MOV #^^IDENT,R3
        MOV #IDENT,R4
.ENDM
```

＝は謂ゆる代入擬似命令で，アセンブル時に実行される．STYLEの定義ではマクロの静的な入れ子が使われている．次にピースLについてのチェック・プログラムを示す．このように書いてみるとデータとプログラムが一体化している様子がよくわかる．

```
402 4024        PIECE   L,N
403 4040        STYLE   PPE,2,2,4,6,50,50,40,40,14
404 4100        STYLE   PPE,1,2,4,6,56,40,30,30,14
405 4140        STYLE   PPI,2,2,4,6,20,30,20,20,14
406 4200        STYLE   PPI,1,2,4,6,12,20,10,10,10
407 4240        STYLE   PPG,1,2,52,122,172,10,10,10,10
408 4300        STYLE   PPF,1,50,120,170,166,60,60,60,14
409 4340        STYLE   PPG,2,50,120,170,172,50,50,50,14
410 4400        STYLE   PPG,1,50,120,170,2,40,40,40,14
411 4440        STYLE   PPH,2,50,120,170,202,30,30,30,14
412 4500        STYLE   PPH,1,50,120,170,12,20,20,20,14
413 4540        STYLE   PPR,1,50,120,170,156,10,10,10,10
414 4600        STYLE   PPH,1,12,62,132,202,30,12,12,12
415 4640        STYLE   PPI,1,12,14,16,20,20,12,12,12
416 4700        STYLE   PPJ,1,12,10,6,4,10,10,10,10
417 4740        STYLE   PPD,1,50,46,44,42,20,20,20,20
418 5000        STYLE   PPE,1,50,52,54,56,10,10,10,10
```

5. 対称解を如何に省くか.

実はこの問題が一番大変であり，最後まで論理的虫（プログラム以前の虫）のとれなかったところである．4で見たようにプログラムの基本的な骨格はかなりスッキリしているが対称解の追放は，かなり無駄な贅肉を呼び込んだようである．IとXに加えてV，さらに必要な場合はW，さらに必要な場合はT（総計5枚）の位置に制限を加えないといけないというのは，いかに立体であるとはいえ大変である．（思うに12個のピースが平面的であることがその最大の原因である．）



左のようにI，X，V，Wを決めても上下の対称解が有り得る（実際にはない．）

対称解を追放する方法としては，まずI，X以外のV，

Wのプリセット（この場合にV，Wのピース・チェックはやらない.）し，それから残った最初のピース（Wあるいは T）のVSPを狭くしてやる方法をとる．ここでは詳しく述べないが基本骨格を変形しないようにするため，全体としてかなり〃とってつけた〃ようなプログラム・セグメントになっている．

6．結果

まずプログラムのサイズについて述べる．アセンブラのステップ数はformattingのための空ステップやマクロを含むので，実際の語数を記す．以下の数値は10進．

種々の初期的擬似命令	50 steps	0 words
初期化＋テーブル	81 steps	430 words
ＥとＸのプリセット	174 steps	1793 words
	（内マクロ定義 111 steps）	
残りのピースのチェック	175 steps	2564 words
PUTOK，BACK，COUNT 等	42 steps	59 words
対称解追放関係	169 steps	1500 words
印刷	53 steps	237 words

と全体としてかなり大きいプログラムであるがステップ数は小さい．

所要時間は印刷をディスクに出して11時間17分30秒ぐらい．ディスクI/Oの時間は約3~4分と思われる．これをラインプリンタに打たせると約30分位かかるようである．解はほぼ10秒に1個の割合で出てくるようであり，10分位の間隔でみるとほとんど一様な速度と言える．

虫取りの都合で文脈チェックの情報を全部10にしたら(すなわち，失敗したら次のスタイルの最初から調べる．)，時間は約1.4~1.5倍かかったので，これ自身は苦労の割には効果があがらなかったらしい．ただ，基底チェックの命令が

```
BIT   R1, the VSP
```

のときと

```
BIT   BITTB-TB(R0), the VSP
```

のとき(毎回マス目番地をたよりにビットパターンをインデックス修飾して持ち出す)とで，1時間時間の差が出たのには驚いた．両者の時間差は2.7μsecだから，このチェックが10億回以上実行されていることがわかる．

結局，端から詰めていくことで約10倍，基底チェックで約2倍，文脈チェックで1.5倍，プログラムのベタ書きで約2倍速度が向上したと私は予想している．ちなみに，実習生の人(プログラム未経験)に作ってもらったプログラムは基本的に磯部氏の方法に近いのであるが一つの解が出るまで4時

間かかった.

7. 展望

当初の予想はなんとか2桁時間達成が目標であったが, 意外に記録が伸びた. 多分5時間位まではいくと予想されるがそのためには相当強力なheuristics が要求されるだろう. ここで述べたプログラムはシラミつぶしであるが非常に高速なTree Traverseを行なっているので, 今後のheuristic programとの比較基準とすることが出来ると思う.

ビット処理による高速化のねらいは, アーキテクチャに多大に依存するが, なかなか思う通りの計算機はないものである.

謝辞

本トリに多大な助言をいただいた池野信一室長, 奥乃博氏に多大な感謝をするものであります.

解の一部（keyboard上に出したもの）

注：印刷プログラムを少しも工夫に作ったので，計算機内でTBのイメージとこの印刷の間には，向きの相異がある．

```
IIIII LLLLN TTTUU   IIIII LLLLN TTTLY   IIIII LLLLN PPPLY   IIIII LLLLN LPPPY
VXFZZ VFFLN VTFFU   VXUUN VFZZN VTPPY   VXUUN VFZZN VPPTY   VXUUN VFZZN VPPTY
XXXZN YYYYN VTYUU   XXXUN FFFZY VTPPY   XXXUN FFFZY VTTTY   XXXUN FFFZY VTTTY
WXZZN WWPPP VWWPP   WXUUN WWFZZ VWWPY   WXUUN WWFZZ VWWTY   WXUUN WWFZZ VWWTY

IIIII LZZPN LLLLY   IIIII UUNNL NNNZY   IIIII YYYYP FFYLP   IIIII LLLLF TTTLF
VXTPN VZTPN VUTUY   VXPPP VUFTL VZZZY   VXTTT VNNNP VFFLP   VXWPF VYWWF VTZWW
XXXPN ZZTPY VUUUY   XXXPP UUFTL VZFYY   XXXTZ NNZZZ VFZLP   XXXPP YYYYF VTZZZ
WXFFN WWTFF VWWFY   WXFTL WWFTL VWWTY   WXUTU WWUUU VWWLL   UXUPP UUUNN VNNNZ

IIIII PPPYW TTTFW   IIIII UTUYL UUUPP   IIIII TTTYL NLLLL   IIIII LLLLN LYYYY
VXZZW VPPYW VTFFF   VXZZW VTZYL VZZPP   WXVFU WTVYU NNVVV   WXUUV WPPPN TTTYF
XXXZW NNNYY VTNNF   XXXWW TTTYL VNNYP   XXXFF WTYYU WNPPP   XXXUV WPPNN WTFFF
UXUZZ UUUYL VLLLL   FXWWL FFFYL VFNNN   ZXFFU ZZZYU WNZPP   ZXUUV ZZZNV WTZFV

IIIII YYYYL NLLLL   IIIII TTTZZ NYYYY   IIIII TTTZZ FVVVN   IIIII ZZTTT NYYYY
WXVFU WYVTU NNVVV   WXVFU WTVZU NNVYP   WXUUU WTUZU FFFVN   WXVFU WZVTU NNVYP
XXXFF WTTTU WNPPP   XXXFF WTZZU WNVPP   XXXPP WTZZN WFYVN   XXXFF WZZTU WNVPP
ZXFFU ZZZTU WNZPP   LXFFU LLLLU WNVPP   LXPPP LLLLN WYYYY   LXFFU LLLLU WNVPP

IIIII ZZVPP NYYYY   IIIII ZZTTT FVVVN   IIIII YYYYT NYUUU   IIIII TTTYZ LYYYY
WXVTF WZVPP NNVUU   WXUUU WZUTU FFFVN   WXFZZ WFFFT NNUFU   WXPPP WTZZZ LLLLV
XXXTF WZZPF WNVUF   XXXPP WZZTN WFYVN   XXXZT WPPPT WNPPT   XXXPP WTZNN WNNNV
LXTTT LLLLF WNVUU   LXPPP LLLLN WYYYY   LXZZV LLLLV WNVVV   FXUUV FFFUV WFUUV
```